COMPUTER VISION

Computer and machine vision is an important field in computer and electronics engineering, overlapping many different skills and disciplines. Most complex animals use vision as their primary source of information about their environment because vision provides some of the richest data of any possible sensor system. Generally, there is too much information contained in the image (e.g., surroundings, lighting, and noise) that is irrelevant to the current application. In addition, images contain much more information about the objects being observed than is necessary. Even a modest camera and frame grabber can achieve resolutions of 640×480. The resulting objects in the image are then composed of hundreds or thousands of pixels. The main focus of computer vision is to extract and reduce the relevant characteristics of the image to a small enough set that the information can be used by other processes, human or machine.

The input to a computer vision system is one or more images, the output of the system describes the objects in the image in the context of a given task. In general, computer vision must be task oriented to reduce the number of possible image interpretations to a reasonable level. For specific images, it is possible to describe the type of preprocessing, extraction, and recognition required. General computers that match the kind of processing exhibited by humans are still far from realization.

An object in the real world does not have a unique description; many descriptions at varying levels of detail and from several points of view can be defined. It is computationally impossible to describe an object completely. Fortunately, we can avoid this potential philosophical snare by considering the task for which the description is intended; we do not want just any description of what is imaged, but one that allows us to take appropriate action.

Computer vision can be divided into two main disciplines, image processing and image understanding. Image processing is concerned with the pixel-level manipulation of images, producing new images with specific information in the image highlighted or suppressed. Image processing is closely linked with signal analysis and discrete mathematics. Image understanding relates conditioned images to models of the world. Image processing works on the pixel level; image understanding works on the image and object level, attempting to match what is observed with what is known. Image understanding is closely linked with pattern matching theory and artificial intelligence.

Image processing maps the original image to a new image, usually in order to highlight some specific information from the original image. These images may have noise suppressed, blurring removed, edges enhanced, contrasts enhanced, thresholding sharpened, or color palettes remapped. Most image processing techniques are based on linear systems theory. Some of the techniques of image processing are useful for understanding the limitations of image formation systems and for designing preprocessing modules for computer vision.

Beyond basic image manipulation, it may be necessary to identify an object or group of objects in an image. Image understanding techniques are used to describe what is observed. Usually the pattern is given as a set of numbers representing measurements of an object, essentially elements or features extracted from the image using image processing. A classifier will generally assign an object to one of a number of classes; this act is usually referred to as object recognition. Researchers concerned with classification have created simple methods for obtaining measurements from images. These methods often treat the image as a two-dimensional array of intensities. Primitives extracted using image processing and or pattern classification algorithms are combined to describe complex objects, and objects are grouped to describe situations. A classical illustration of scene analysis is the interpretation of line drawings. In this case, the image is described as a set of polygons, which in turn are given as a collection of line segments. Before these descriptions can be used for a task, the topology must be discovered—specifically, which regions bounded by the lines form objects. It is also important to know how objects relate to one another. Scene analysis therefore extracts a meaningful symbolic representation from image data.

The generation of descriptions from images can often be conveniently broken down into two stages. The first stage produces a sketch, a detailed but undigested description. Later stages produce more parsimonious, structured descriptions suitable for decision making. Processing in the first stage will be referred to as image processing, whereas subsequent processing of the results will be called image understanding.

FRAME GRABBERS

Visual images come in a variety of types such as a three-dimensional scene captured into two dimensions by a video camera and stored on a tape medium. Images can also originate outside of the visual electromagnetic spectrum—encompassing ultraviolet light, X rays, and radio waves (1). Such images can be generated by radio astronomy as perceived by a radio telescope. On the acoustic end of the frequency scale, ultrasonic signals are used to generate ultrasonic images of the fetus within a mother's womb. Subacoustic signals such as sonar are used to map out features deep within the ocean and within the earth's crust.

Regardless of the source or type of image, image processing requires the acquisition of the visual-based information and transfers it into the computer's memory or other storage medium. The process of acquisition is most-often achieved with frame-grabber hardware. As the name indicates, frame grabbers capture or "grab" individual frames of a video signal into the computer's memory—converting the video signal through digitization into a format that can be used by the computer hardware and software (2). This format constructs the image as a two-dimensional array of picture elements, or pixels, with each pixel associated with a brightness or color value. Computer processing of the image operates on these pixels, which constitute the picture. Although images come from a staggering variety of sources, from a visual scene from a CCD camera to an X-ray image generated from a scanning-electron microscope, by far the most popular means of transmitting these images is via standard analog video signals. Hence, the majority of image acquisition hardware in use today consists of digitizing frame-grabber hardware that retrieves such video signal information into the computer (3).

The process of digitizing video is relatively straightforward but is complicated by the standardized video signal formats. Monochromatic composite video used in North American television is perhaps the most common video signal format employed to encode brightness or luminance images through a single wire. Also known as the RS-170 video signal convention, brightness information is encoded as a voltage varying over a 0.7 V range as referenced from the black level shown in Fig. 1.

The horizontal sync pulses separate each horizontal scan line into 63.5 μs intervals with a full video frame consisting of 525 such scan lines. Each frame, however, is interlaced into odd and even horizontal scan lines resulting in a complete frame every 1/30th of a second.

Digitization of the composite video signal requires analog-to-digital conversion (ADC) hardware as well as circuitry to clock the ADC properly and to account for the vertical retrace that signals the end of a interlaced frame. Clock circuitry subdivides each scan line into the horizontal resolution of the final grabbed image—typically 512 or 640 pixels. The number of usable scan lines counted by the frame grabber determines the vertical resolution of the final image. Although it is conceivable to use a 525 pixel vertical resolution, a 512 or 480 pixel resolution is usually chosen because of technical limitations or for hardware simplicity. A 480 pixel vertical resolution is commonplace because, out of the 525 horizontal scan lines in a complete frame, only 480 are typically usable in reality (1).

With proper clocking, typical frame grabbers employ a flash ADC to convert the time-varying analog voltage signal to a stream of bit values of intensity associated with the discrete pixels that make up a horizontal scan line. The process is illustrated in Fig. 2.

Each digitized pixel value is between 0 and 255, where 0 represents black and 255 represents white in the image. Although high-resolution ADC hardware can be used to increase

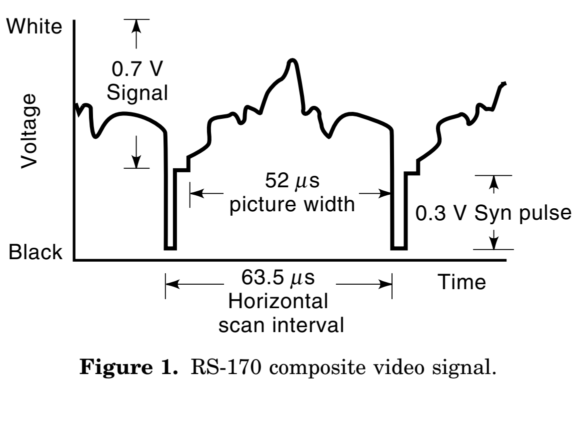


Figure 1. RS-170 composite video signal.

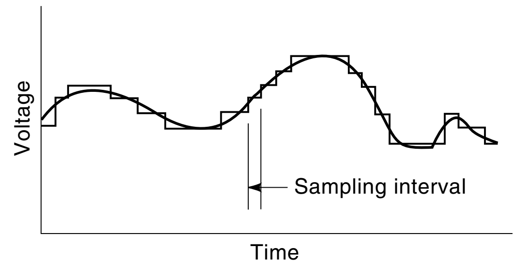


Figure 2. Analog-to-digital conversion of a scan line.

the number of possible intensity values, bandwidth limitations imposed on the composite video signal used in broadcast television effectively constrain the number of discernible voltage values to around 330. Complete acquisition of a video frame results in a digitized image that is usually either 512 $\times$ 512 or 640 $\times$ 480 pixels in size. The latter image size is preferable because its width-to-height ratio matches the standard 4 : 3 aspect ratio of composite video frames. Acquired images thus exhibit minimal spatial distortion.

The introduction of color into the imaging equation increases the complexity of the acquisition process. In addition to brightness or luminance information, color or chrominance information is detected and transmitted with the image. In the composite color video format, chrominance and luminance signals are combined into a single signal for transmission through one wire in the NTSC color-encoding scheme. Its time-varying signal is practically identical to the monochromatic composite signal for compatibility with broadcast television. This scheme packs additional color information into the already narrow bandwidth of the signal, resulting in both poor color reproduction and inferior spatial resolution. By transmitting the luminance and chrominance on separate signals, spatial image quality is increased. This method is used by the S-video or Y-C video format found in Hi-8 and S-VHS video recorder systems. Some imaging systems also provide separate red, green, and blue composite video signals for digitization, which result in improved color reproduction (1).

A number of schemes exist for encoding color into an analog video signal. Nevertheless, the techniques for digitizing the color image information through a frame grabber remain more or less the same. In the case of NTSC color encoding, the single composite signal can be digitized in an identical manner as in the case of the monochromatic composite signal. Color information, however, will need to be decoded by additional hardware. In the case of the multiple color signal formats, digitizing the video is even simpler. The multiple signals, such as the luminance and chrominance, are merely monochromatic composite signals that encode the different components of color. As such, they can be digitized by the previously described frame-grabber hardware via parallel ADCs.

The use of standard analog video in both monochromatic and color image transmission certainly lends compatibility and flexibility to its use. For technical applications, however, such flexibility constrains the resolution and accuracy of the acquired images within the limited bandwidth of the analog video signal. More rigorous applications, such as astronomy, produce high-resolution images with luminance ranges that exceed the encoding ability of standard analog video. Such specialized applications require high-end frame grabbers that can digitize 16 to 24 bits of luminance range. Some imaging systems, in fact, forgo the conversion of image information to video for transmission and its subsequent digitization by delivering the image as a digital stream from the source. This process reduces the noise introduced from both signal conversions and the complexity of transmission and digitization hardware.

IMAGE PROCESSING

Image processing is generally the first step of a computer vision algorithm. Image processing performs preprocessing on the image to highlight specific features, eliminate noise, and gain basic pixel-level information about the image. Image processing is used to reduce the amount of information in an image from raw pixel data to a more cohesive subset that the image understanding algorithms can use. Image processing techniques are closely tied with digital signal processing and signal analysis theory.

The Histogram

One of the simplest operations that can be performed on an object is the computation of its gray-level histogram. The gray-level histogram of an image is a graphic representation of the distribution of intensity values within the image. The histogram is created by plotting the number of pixels in the image corresponding to each possible gray-scale value. The histogram can provide valuable information on the composition of the image. An image with good contrast will have several peaks in the histogram with pixel gray-scale levels covering the entire possible intensity range. The brightness of an image can be determined by examining the number of pixels at the high and low end of the spectrum. If the image is composed of a single object on a background, as is often found in industrial settings, the histogram can illustrate the gray level corresponding to the best threshold (3).

Even though the histogram as a visual construct is not used directly in image processing, the data of the distribution of the gray-scale values can give valuable information on the image for automatic contrast and brightness adjustment. The derivative of the histogram can give valuable information on the gray-scale boundaries for objects in the image, which can be used to determine thresholding levels for segmentation.

An example of an image and its histogram is shown in Fig. 3. The histogram can be analyzed to determine the level of contrast in an image. If the image is not ideally contrasted as the image shown in Fig. 3, a balanced histogram can be created automatically. This process is known as histogram equalization. In general, histogram equalization attempts to fit a constant distribution to the gray-scale image, which means that the total occupancy of the gray-scale values increases linearly. Fitting this form of distribution to the histogram can be expressed by using the following equation:

$$i_{\mathrm{n}} = \frac{i_{\mathrm{h,n}} - i_{\mathrm{l,n}}}{i_{\mathrm{h,o}} - i_{\mathrm{l,o}}}(i_{\mathrm{o}} - i_{\mathrm{h,o}}) + i_{\mathrm{l,n}} \tag{1}$$

where i_{h} is the highest intensity value, i_{l} is the low intensity value, i_{o} is an intensity in the previous distribution, and i_{n} is an intensity in the new distribution (4). It should be apparent that Eq. (1) is derived from the linear equation of the total gray-scale distribution. An example of histogram equalization is shown in Fig. 3.

Thresholding

Thresholding is used to reduce the number of gray scales used to represent the image. The reduction of gray-scale values can often eliminate extraneous information or noise from an image. A thresholded image is usually reduced to a binary form (black and white) to accentuate certain features and reduce the computation required for scene analysis. Thresholding is often based on the intensity or gray-scale histogram of the image (1).

The simplest thresholding operation is mapping an image from a gray-scale image to a binary image. The image is scanned left to right and top to bottom. Any pixel with a gray-scale value greater than the threshold is changed to white, and any pixel with a gray-scale value less than the threshold is changed to black, as shown in Eq. (2). A more general version of the thresholding operation is shown in Eq. (3) (2). The operation will turn a pixel white if it belongs to set F and black if it does not. This type of threshold is useful for applications where the image may contain more than one gray scale:

$$I_{ij} = \begin{cases} 1 & \text{if} I_{ij} > T \\ 0 & \text{if} I_{ij} < T \end{cases} \tag{2}$$

and

$$I_{ij} = \begin{cases} 1 & \text{if} I_{ij} \in F \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

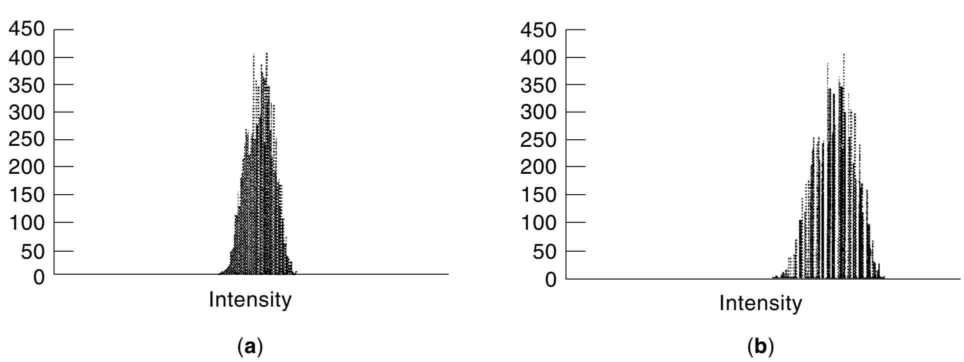
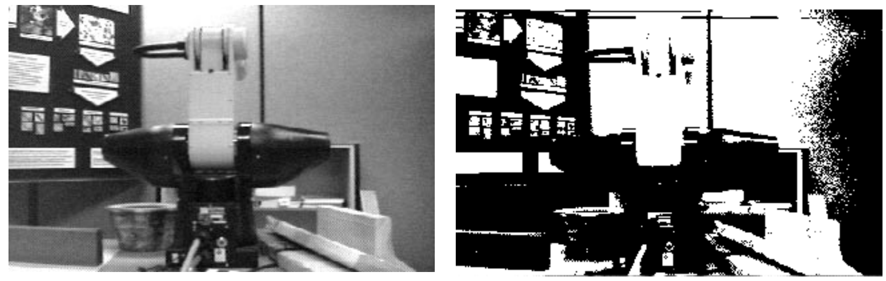


Figure 3. Example of histogram equalization: (a) original image of lab, (b) equalized image.



(a) (b)

Figure 4. Example of thresholding: (a) original image, (b) thresholded image with $T = 128$.

The thresholding operation is useful in a number of situations. If an object has a high degree of contrast with the background, a thresholding operation can effectively segment an image. Thresholding can also be used to remove noise from an image after an edge detection algorithm has been employed. An example of binary thresholding is shown in Fig. 4.

Often the exact gray-scale value of the threshold cannot be determined beforehand and must be determined by the computer. This is known as adaptive thresholding. There are many adaptive thresholding techniques; the common techniques are highlighted here. Perhaps the most common technique is a valley-finding approach based on the histogram. If the image has a bimodal (two-peaked) histogram, as a dark object on a light background would have, then the best location for the threshold is between the two peaks (5). However, this is often difficult to determine because noise can create several local peaks and valleys, and it becomes necessary to determine the "peakedness" of each peak before assigning a threshold. The situation becomes much more complicated for multimodal peaks because aliasing from neighboring distributions may make the peak almost impossible to detect.

Texture-Based Thresholding. A texture is simply a repeated pattern of pixels over the surface of an object. Textures can be caused by surface characteristics such as painted polka dots or geometric characteristics such as crystal structure in metals. Texture detection is used where the gray levels are the same, and distinct edges are difficult to locate because the edge detector will be limited by the small high-gradient areas in the image due to the texture itself (3). Textures can be a useful tool in highlighting different regions of the system; however, they can also be very difficult to detect robustly. Textures must be detected using statistical methods that take both spatial and gray-scale levels into account because texture is a spatial phenomenon. Textures are decomposed into texture primitive of texels, which describe the texture in question. The texel is one of the repeated elements in the pattern, and these texels can be combined into complex textures using a cell and array grammar (1). Measurements such as entropy and autocorrelation can describe the appearance and periodicity of a texture (2).

Filters

Filters are used in image analysis to perform pixel-level adjustments. Common applications of filtering include noise reduction and edge enhancement. Filtering can also be used to adjust an image to cut down glare or other undesirable effects, making the image easier or more appealing to examine. Filtering is generally the first step in automated image processing.

Image filtering is based on ideas borrowed from digital signal analysis. Specifically, the signal (image) is convoluted with an ideal digital filter to suppress some characteristics and accentuate others. An image filter is a matrix instead of a vector as used in digital signal analysis, because much of the image's information is stored in two-dimensional relationships. The matrix that encodes the filter is generally called a kernel. The kernel is generally 3×3 to reduce computational intensity, but larger kernels are possible. Figure 5 shows a kernel and its matrix representation.

For image processing operations the current pixel is the center element of the matrix. The result of the filtering operation for pixel$_{xy}$ having intensity I_{xy} is the dot product of the matrix and the neighbors of pixel$_{xy}$ (5):

$$\begin{aligned} I_{xy}(n+1) = {} & aI_{x-1,y-1} + bI_{x,y-1} + cI_{x+1,y-1} + dI_{x-1,y} + eI_{x,y} \\ & + fI_{x+1,y} + gI_{x-1,y+1} + hI_{x,y+1} + iI_{x+1,y+1} \end{aligned}$$

The effect of the filter is changed by varying the coefficients. The image is filtered when the mask has been convoluted with every pixel in the image. This is in essence an approximation of the discrete convolution given by

$$h[i,j] = f[i,j]^{*}g[i,j] = \sum_{k=0}^{n} \sum_{l=1}^{m} f[k,l]g[i-k, j-l] \qquad (4)$$

Most filters are linear and therefore can use the convolution equation directly; however, there are exceptions like the me-

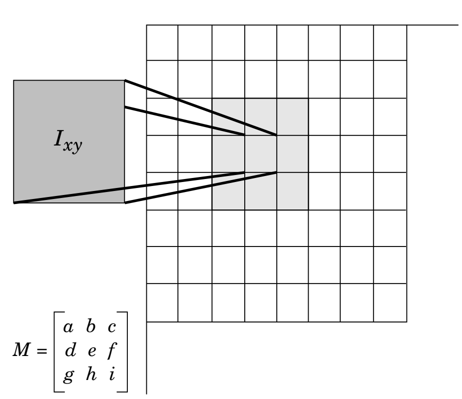


Figure 5. Example of a 3×3 neighborhood and its matrix representation.

dian filter and the Canny edge detector, which are nonlinear operations, usually implemented as step-by-step algorithms.

In general, there are two major classes of filters: smoothers and enhancers. The smoothing filters are generally two-dimensional versions of low-pass filters. They are used to eliminate noise and rough edges caused by sampling. Enhancers belong to two classes: contrast enhancers and edge detectors. Both accentuate high-frequency information, but contrast enhancers work on the entire image, whereas edge detectors only detect regions of very high-intensity gradients. Types of filters are explained in more detail in the following sections (4).

Smoothing Filters. Smoothing filters remove noise and rough edges by approximating low-pass filters. The three most common smoothing filters are mean, median, and Gaussian. Mean and Gaussian filters are linear and obey the general convolution laws described previously. The median filter is not linear but is easy to implement using similar ideas to those already developed. There is a trade-off in using smoothing filters to eliminate noise: smoothing can eliminate noise, but it also blurs the image, obscuring edges and making object detection less accurate (3).

Mean Filters. The mean filter does precisely what its name implies, it takes the mean of a set of values. In this case, the mean filter changes the value of the pixel to the average of the pixel and its neighbors. This filter is equivalent to a moving average filter in traditional signal analysis. Like a moving average filter, it is not a perfect low-pass filter, but it is very effective at reducing the amount of noise in a image. Mean filters can be normal mean filters or weighted mean filters. In a normal mean filter, the value of the each cell is one [or 1/(number of cells) for floating point]. In a weighted mean filter, some of the cells, usually the center cell (which corresponds to the center pixel) are given a higher proportional weight. Examples of kernels follow (5):

$$\text{normal}: \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \qquad \text{weighted}: \begin{bmatrix} 1 & 1 & 1 \\ 1 & c & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{5}$$

In practice, mean filtering is generally a two-step process. Because images are usually encoded as arrays of integers, it is preferable to use integer math rather than floating-point math. Even though mean filtering can be accomplished in a single step using rational values for the cells, it can be faster to use integral cell values as shown and divide by the number of cells, thus getting an average. The true 3×3 mean filtering operation would be

$$I_{xy} = \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} I_{ij} \tag{6}$$

where I_{ij} is the matrix or window containing the current pixel and its neighbors. Although mean filtering can be an effective method of removing noise, it has the undesirable side effect of blurring edges by reducing the intensity gradient in the region of the edge. The result is weaker edges when passed through an edge-detecting filter. The degree of filtering and therefore smearing is determined by the size of the matrix. Even though 3×3 is the most common, larger matrices with greater filtering effects can be used to eliminate noise and remove partial edges.

Median Filters. The median filter is actually not a true linear filter; however, it is convenient to think of it as one, and it will be treated with the rest of the filters. The median filter is actually a simple algorithm. The median algorithm simply changes the current pixel intensity to the median of the intensity of the surrounding pixels. Median filtering is very effective at removing salt and pepper noise, isolated spots of noise in an otherwise uniform field of view. A mathematical interpretation of the median filter follows:

$$I_{xy} = \hat{I}/I \in N$$

where I_{xy} is the current pixel intensity and N is the neighborhood of I. Like the mean filter the value of N can be increased to alter the amount of filtering, although the result is less dramatic than with the mean filter.

Gaussian Filtering. A Gaussian filter is generally used to eliminate Gaussian noise. The Gaussian filter is one of the most powerful filtering types because of the characteristics of the Gaussian function. The advantages of Gaussian filtering follow:

1. Gaussian functions are rotationally symmetric.
2. Gaussian weighting decreases with distance from the current pixel.
3. The Fourier transform of a Gaussian function is a Gaussian function.
4. The degree of smoothing can be altered with by changing a single parameter.
5. Gaussians are separable and can be decomposed into horizontal and vertical components (6).

A discrete two-dimensional Gaussian filter is given by the following equation:

$$g[i, j] = e^{-(i^2+j^2)/2\sigma^2} \tag{7}$$

where i and j are the coordinates of the current pixel and σ is the width of the function (5). The size of σ determines the degree of smoothing by changing the shape of the Gaussian generated. A larger σ results in a flatter Gaussian with more even weightings. As the Gaussian approaches a square function, the filter performs more like a mean filter (6).

Gaussian functions are important for filtering applications because they yield a weighted average where the weightings drop monotonically with distance from the current pixel. This behavior reduces the amount of edge blurring because pixels at the edge of a gradient are given less weight than pixels at the center of the gradient as the filter passes over the gradient. In addition, Gaussians can be separated into horizontal and vertical components. This saves computational cycles because the filter can be implemented as two single-dimensional convolutions instead of a single two-dimensional convolution.

Enhancement. Sharpening is a contrast-enhancing filter that behaves like a high-pass filter. Like most types of high-

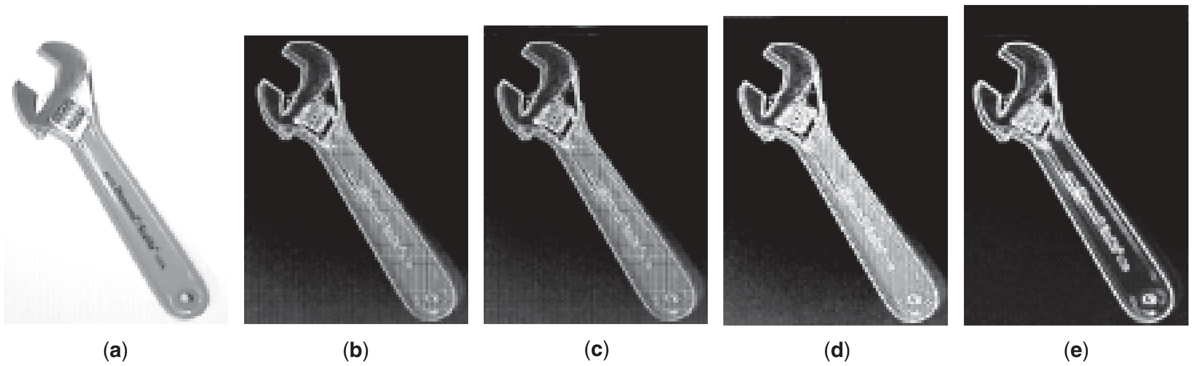


Figure 6. Examples of gradient edge detection: (a) original image, (b) vertical Roberts filter, (c) horizontal Roberts filter, (d) Sobel edge enhancement $c = 2$, (e) Sobel edge enhancement $c = 2$, prefiltered with a mean (moving average) filter.

pass filters, a sharpening operation is very sensitive to noise. Sharpening filters can be used to accentuate portions of the imagelike boundary regions (5).

$$\text{type 1} \quad \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} \quad \text{and type 2} \quad \begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix} \tag{8}$$

Edge Detection

Image analysis is the exercise of extracting a small precise description of an image from a large number of pixel elements. One of the simplest ways to accentuate images is to extract edge information from the image. Edges can be used to discover external and internal boundaries of objects and can, consequently, be used for image segmentation, feature extraction, and object recognition. Edge detection is often one of the first steps employed with image analysis.

Edge detection is usually performed by detecting rapid changes in intensity. Geometric edges of objects or features usually correspond to areas in the image containing rapid changes in intensity. There are several ways to detect the change in intensity, but most are based on examining the local derivatives of the image. In most edge detection algorithms, the edge is detected by convoluting the image with a kernel designed to highlight the edges of the image, resulting in a new image that contains only detected edges.

Because most edge detection algorithms use derivatives to detect the edges, they are sensitive to noise. To compensate, edge detection is often accompanied by filtering or thresholding operations to eliminate noise in the image before or after filtering. As with most low-pass filtering, noise reduction comes at the cost of sensitivity. The more noise eliminated, the less distinct the edges will become. See Fig. 6 for examples of edge detection (3).

Gradient-Based Edge Detection. The most common method for detecting edges in an intensity map is to use a gradient enhancer. The gradient enhancer is a special filtering process that highlights edges by returning the magnitude of the gradient of the image. Edges tend to be characterized by rapid changes in intensity and have high local gradients. Continuous surfaces generally have uniform intensity levels and small local gradients.

Gradients are generally approximated from pixel data using the difference operator. Because the gradient is being computed for a surface, the gradient has two components, an x component and a y component. The magnitude of each gradient is computed using the difference operator in a local region. Table 1 shows the three most common edge detectors, their convolution kernels, and the equivalent algebraic expression for the operation. All operators are illustrated with 3×3 kernels, although larger kernels are possible (4).

The x component of the gradient G_x calculates the horizontal slope and tends to enhance vertical edges. The y component of the gradient G_y calculated the vertical slope and tends to enhance horizontal edges. The kernels will actually highlight all edges, but the degree of enhancement depends on the direction of the edge. Like any gradient, the intensity gradient can be expressed as a magnitude and direction. The magnitude and direction of the gradient are given by (5)

$$G_m = \sqrt{G_x^2 + G_y^2} \qquad G_\theta = \tan^{-1}\left(\frac{G_y}{G_x}\right) \tag{9}$$

Laplacian Edge Detectors. Even though edge detection using the gradient is simple and the most commonly used, it often does not highlight edges enough for identification. Using the second derivative of the intensity function, or the Laplacian, it is possible to detect weaker edges. However, the Laplacian, like most second derivatives, is very sensitive to noise. Edge detection with the Laplacian can result in many extraneous edges.

To eliminate the extraneous edges, prefiltering is often used, as in the Canny edge detector. In the Canny edge detector, the image is filtered with a Gaussian filter; then a gradient or Laplacian edge detector is applied, and the result is thresholded to highlight the desired edges. A special case of this operation is the Laplacian of Gaussian approach, where the second derivative of a Gaussian function is computed and used as the convolution kernel to detect edges directly. The Laplacian of Gaussian approach is valid because the separate convolution of Gaussian and Laplacian operations are commutative and combinable. The resulting function is a Mexican hat function that can be approximated with a single kernel. The Laplacian of Gaussian function provides both filtering

Table 1. Examples of Edge Detection Operators

Filter Name	G_x Kernel	G_y Kernel	Algebraic Equivalent	Remarks
Roberts	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & -1 & 1 \\ 0 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 1 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	$G_x = I_{x+1} - I_x$ $G_y = I_{y+1} - I_y$	The most basic operator, which takes a very simple approximation of the gradient.
Sobel	$\begin{bmatrix} -1 & 0 & 1 \\ -c & 0 & c \\ -1 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} -1 & -c & -1 \\ 0 & 0 & 0 \\ 1 & c & 1 \end{bmatrix}$	$G_x = I_{02} + cI_{12} + I_{22} - I_{00} - cI_{10} - I_{20}$ $G_y = I_{20} + cI_{21} + I_{22} - I_{00} - cI_{21} - I_{22}$	The most common edge detector. I computes the gradient with a weighted sum. The pixels nearest the current (center) pixel are weighted higher.
Prewitt	$\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$	$G_x = I_{02} + I_{12} + I_{22} - I_{00} - I_{10} - I_{20}$ $G_y = I_{20} + I_{21} + I_{22} - I_{00} - I_{21} - I_{22}$	Special case of the Sobel filter with $c = 1$.

and edge detection in a single filter, saving computational cycles over the multiple-step Canny edge detector (4).

Like all other forms of filtering, Laplacian edge detection relies on the convolution of kernels that approximate the Laplacian. The kernels are iteratively convoluted with the local pixels of the current (center) pixel to achieve the edge filtering. Table 2 contains examples of kernels used for Laplacian edge detection.

Because the Laplacian operators are very sensitive to noise, they are not as common as gradient operators. The noisy behavior of the operators can be seen in Fig. 7.

Contour and Shape Detection

Even though edge detection can isolate regions of high gradient intensity, the image is still expressed as a matrix of pixels, and additional processing is required to express the image in a usable format. The image must be analyzed again to isolate higher-order shapes and edges. After an edge detection operator has been applied to the image, the detected edges must be expressed as lines, arcs, or shapes. Only after an image has been reduced to geometric primitives can meaningful shape-based analysis be performed. Most commonly, edges are approximated by a sequence of linear segments, although in special cases other curves or even complex shapes can be used (3).

There are three basic types of contour detection: mask matching, graph searching, and voting mechanisms. In mask matching, a primitive of a known object is passed over the image. If a match between the shape of the primitive and a shape within the image is found, then the contour of the object is expressed as the primitive. Graph searching is a general computing technique found in many applications. In image processing, the algorithm starts with a point known to be

Table 2. Examples of Laplacian Edge Detectors

Filter Name	Kernel	Derivation	Remarks
Laplacian 4	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	$\frac{\partial I}{\partial x} = I_{x,y+1} - 2I_{x,y} + I_{x,y-1}$ $\frac{\partial I}{\partial x} = I_{x+1,y} - 2I_{x,y} + I_{x-1,y}$	Approximation of the second derivative of the intensity. Susceptible to noise.
Laplacian 20	$\begin{bmatrix} 1 & 4 & 1 \\ 4 & -20 & 4 \\ 1 & 4 & 1 \end{bmatrix}$	$\frac{\partial I}{\partial x} = I_{x,y+1} - 2I_{x,y} + I_{x,y-1}$ $\frac{\partial I}{\partial x} = I_{x+1,y} - 2I_{x,y} + I_{x-1,y}$	Approximation of the second derivative of the intensity. More weight placed on the center pixel than in Laplacian 4.
Laplacian of Gaussian	$\begin{bmatrix} 0 & 0 & -1 & 0 & 0 \\ 0 & -1 & -2 & -1 & 0 \\ -1 & -2 & 16 & -2 & -1 \\ 0 & -1 & -2 & -1 & 0 \\ 0 & 0 & -1 & 0 & 0 \end{bmatrix}$	Discrete approximation of Mexican hat function: $\nabla^2 = \left(\frac{x^2 + y^2 - 2\sigma^2}{\sigma^4}\right) e^{\frac{-(x^2+y^2)}{2\sigma^2}}$	Approximation of the second derivative of the Gaussian function. Performs edge detection and smoothing simultaneously.

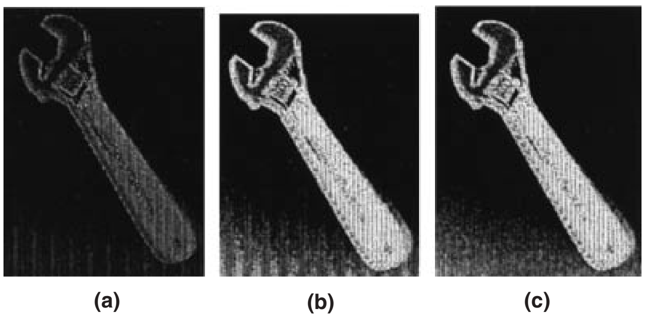


Figure 7. Frame (a) contains the results of a Laplacian 4 filter. Frame (b) contains the results of a Laplacian 8 filter. Even though the Laplacian 8 gave better results, it was also the most sensitive to noise. Frame (c) contains the results of a Laplacian of Gaussian filter.

on an edge and searches iteratively through all adjoining pixels until it has found all the pixels on an edge. The algorithm then uses a mathematical process such as least mean squares or splining to fit a curve to the given set of points. The final method takes a set of points known to form an edge and finds the optimal fit by paramatizing the pixel set into shape space, and finding the shape that best describes the set. This method is called a voting mechanism. The most commonly used voting mechanism in image processing is the Hough transform (3).

Masking. One of the simplest algorithms for detecting a specific contour in an image is masking. In this step a mask is passed across an image, and the difference is computed. When the difference is sufficiently close to zero, the mask and image match, and the contour of the group of pixels is known to be the mask. This method is not very general but can be very powerful if there is a finite set of expected shapes. Consider the example where the purpose of the vision system is to recognize the numerical codes printed on the bottoms of cheques. There are a finite number of possible masks—the digits 0–9 and some assorted special bar codes. The digits will always appear in exactly the same place and with the same orientation so the masking search is quite reliable (2).

Graph Following. Graph-following algorithms use simple tree-search-like steps to find the best digital or geometric representation of an edge. Graph following is more properly an intermediate step where regions of an image likely to belong to the same contour are isolated. The pixels are stored in an array and processed again using one of many traditional line- or curve-fitting techniques. A graph-following algorithm is usually executed after an edge detection algorithm. The graph-following algorithm starts at a pixel known to be on a contour (usually just the gray-scale value) and adds pixels to the edge list based on a heuristic. The heuristic may be simple (e.g., "add all pixels with an intensity greater than I_T") or complex (e.g., "add all pixels such that the curvature of the resulting line is minimized").

Graph following can have two results—detecting the member pixels of a curve for interpretation using a curve fitting algorithm or storing a minimum number of points along a boundary in order to perform discrete segmentation and feature extraction. The complexity of the evaluation function depends on the use of the data. If data are being collected for curve fitting, the algorithm is generally somewhat simpler because the curve-fitting algorithm should select the best fit for the curve. If the purpose of the graph-following algorithm is for discrete boundary isolation, then the evaluation algorithm is more complex because the graph-following algorithm must determine the best representation of the curve itself (3).

If the purpose of the graph search is to create a discrete best-curve description, the system should use a form of tree search. All possible paths through the pixels should be computed and assigned a cost function. This cost function could be based on line length, curvature, intensity, or many other metrics or combinations of metrics. The search can be performed iteratively, combining steps like

1. Find all pixels with intensity $> I_T$.
2. Find the path from the preceding result that minimizes curvature.

In this example, the search is global; that is, all possible paths are searched, and the lowest curvature path is selected. However, this can be computationally intensive for long paths with wide gradients. Often it is preferable to trim branches of the path that are not promising, limiting the search. The search may not reach a global minimum, but the computational load would be much less. Typical tree-search algorithms found in computer science are

- Tree pruning,
- Depth-first searches,
- Modified heuristic searches.

If the purpose of the graph following is merely to select a set of data points for subsequent curve fitting, the requirements on the graph search are much less stringent and generally involve the gradient intensity and direction. After a set of points has been selected, one of several curve-fitting algorithms can be employed (3).

Traditional Best-Fit Algorithms. Algorithms that fit a curve to a set of points have existed for hundreds of years. Although traditional best-fit algorithms usually deal with a functional approximation for a given set of data, they can be extended to computer vision. If the pixels along an edge are considered data points, it is possible to find a curve that approximates them. Examples of traditional curve-fitting algorithms are least mean squares, linear interpolation, and splining. For brevity, we will deal only with fitting lines to the data.

Depending on the required accuracy of the representation, all the points generated by the search algorithm may not be required to obtain an accurate geometric description of the contour. If a rough approximation is sufficient, an algorithm can be used to follow the inside or outside the thresholded gradient of a detected edge. Two examples of linear approximation algorithms are the split algorithm and the merge algorithm.

In the merge algorithm, the contour starts at a given point and is followed until the break condition is reached. When the break condition is reached, a line segment is calculated from the start point to the break point; then a new line is started, and the contour tracing continues until the end point is reached. At the end, another pass can be made over the line segments, merging together adjacent line segments that are similar in slope.

In the split algorithm, a line segment is drawn between the start and end points of the contour. The contour is then traced until a break condition is reached (e.g., if the curve is a specific distance from the line segment). When the break condition is reached, the original line segment is broken, so there are two line segments connected at the break point and end points. The process continues until the entire curve has been traced. The split algorithm is very similar to the merge algorithm only it operates from global to specific instead of specific to global. An example of split-and-merge algorithms is shown in Fig. 8.

The Hough Transform. The Hough transform is another method for finding the equation of a line through a set of points. The Hough transform deals with the line in parameter space instead of variable space. The equation is expressed as a set of coefficients instead of points. The Hough transform essentially considers all possible lines, and picks the one that best fits the data. For example, the Hough transform for a line is

$$y = mx + b \qquad \text{Linear equation} \tag{10}$$

$$b = y - mx \qquad \text{Hough transform} \tag{11}$$

In the first equation, the x and y are variables, and m and b are constant. In the Hough transform, b and m are variables, and x and y are constant. A set of lines in Hough space is chosen and assigned accumulators. After the parameter space has been quantized, every point in the set believed to be on the line is transformed into Hough space. Every line that satisfies the current point is incremented by 1. When all points have been analyzed, the accumulator array can be examined. Peaks in the accumulator array correspond to good-fit lines for the data (3).

Morphology

Mathematical morphology can provide an extensive set of tools for analyzing an image. Morphological operations occupy the same image processing niche as filtering operations. They are generally used as a preprocessing tool to condition an image for easier recognition. The major difference between filtering and morphological operations is that filtering changes the image based on frequency characteristics, and morphological operations change the object on the basis of shape characteristics. Because morphological operations are based on shape, they are useful if there is a good deal of a priori knowledge about the expected shape of the object.

The morphological approach has the additional advantage of mathematical rigor. Instead of each image being expressed as the result of a convolution filter as with the frequency case, morphological operations are built from primitive operations and allow the user to express imaging processing as a kind of algebra.

Morphological operations are based on set theory, where the main set (the image) is compared with a subset to determine values such as shape. Morphological operations are performed on binary images, but they can be performed on gray-scale images as well. Basic set operations for binary and gray-scale images are defined in Table 3 (7).

All morphological operations are built from the preceding set operations. The remainder of this article will deal only with the binary (black-and-white) versions of morphological operations, but in general the processes are valid for gray-scale images using the preceding primary relations.

Translation-Based Morphological Operators. Translation-based morphological operations can be described in two ways—rigorously using the definitions from set theory or qualitatively describing how they affect an image. Qualitatively, translation-based morphological operations work as local neighborhood operators, changing the state of a pixel based on the state of its neighbors. Functionally, this is similar to the median filter where the value of the current pixel is changed to the median gray-scale value of the neighboring pixels. Morphological operations are slightly more sophisticated because they allow the user to define those pixels in the vicinity of the current pixel.

Mathematically, a translation-based morphological operation is defined as the union or intersection of a pixel with a set of surrounding pixels defined by a structuring element or stelt. The structuring element is an array of vectors describ-

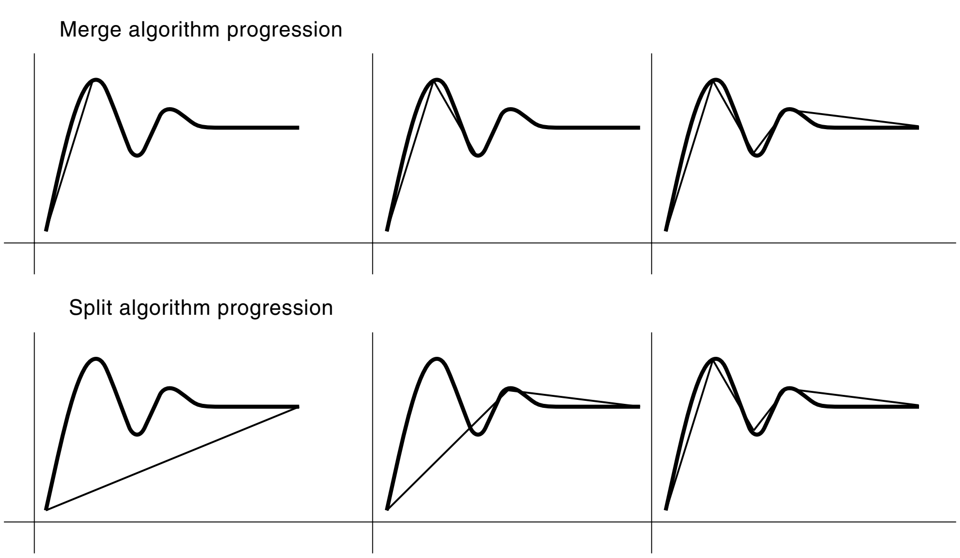


Figure 8. Examples of linear interpolation algorithms. Top: merge algorithm creates an interpolation from front to end. Bottom: split algorithm creates an interpolation by continually dividing the original interpolation.

Table 3. Examples of Morphological Operations

Operator	Expression	Binary Evaluation	Gray-scale Evaluation
Union	A ∪ B	C = A OR B	C = max(A,B)
Intersection	A ∩ B	C = A AND B	C = min(A,B)
Complement	A′	C = not A	C = (MAXVAL − A)

ing which pixels relative to the current pixel are relevant for the current computation. A structuring element can also be viewed as an image or matrix. If the structuring element is viewed as a matrix, then every element in the matrix that is 1 (or TRUE) corresponds to a relevant pixel in the list of vectors. The image/matrix stelt must have a defined origin, usually the center of the stelt, corresponding to the current pixel in question. Although the image and matrix representations are the simplest to understand, the vector list is the most computationally efficient. An example showing the equivalence of the array, matrix, and image representations of a stelt is shown in Fig. 9.

The selection of stelt determines the behavior morphological operations. The stelt can be used as a probe to determine the type of shape that the current image has, or it can be used to accentuate certain features in the image based on its shape (7).

Erosion and Dilation. A stelt by itself is useless; the stelt must be defined over an operation. The simplest morphological operations are erosion and dilation, which are, respectively, defined as the intersection and union of the current pixel with the stelt. If the union between the current pixel and a pixel in the stelt evaluates to true, then the dilation is true, and the current pixel is set to one. If the intersection between the current pixel and every pixel defined by the stelt is true, then the erosion evaluates to true. Erosion and dilation are expressed mathematically (2):

$$\text{Dilation} \quad A \oplus B = \bigcup_{\forall b_i \in B} A_{b_i} \tag{12}$$

$$\text{Erosion} \quad A \ominus B = \bigcap_{\forall b_i \in B} A_{bi} \tag{13}$$

Dilation as an operation has a tendency to expand (dilate) the foreground of the image, filling in holes and accentuating noise. Erosion has the tendency to thin (erode) objects in the image, enlarging holes but removing noise. Dilation and erosion can be used with specific stelts to probe the shape of an object as described previously. Erosion can be used to see if the object in the image has areas with the same shape characteristics as the stelt.

Dilation and erosion can be iterated on an image if several levels of expansion or reduction are required to fill holes or eliminate noise. However, the more iterations performed, the more the object begins to resemble the stelt. If too many iterations are performed, valuable feature information about the object may be lost (7).

Opening and Closing. Dilation and erosion also form the basis for most of the more complex morphological operations used in image processing. Sequential iterations of erosions and dilations can produce much more powerful operators for image analysis. The most common types of operations performed using combinations of erosions and dilations are opening and closing. Opening is the operation that results from performing one or more erosions, followed by an equal number of dilations using the same stelt. Closing is the operation that results from one or more dilations followed by an equal number of erosions using the same stelt. As the names suggest, opening tends to accentuate convex regions of the image, whereas closing tends to eliminate them. Mathematical definitions of opening and closing are shown in the following equations (3).

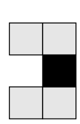
$$(A \circledcirc B) = ((A \ominus B) \oplus B) \qquad \text{Opening} \tag{14}$$

$$(A \odot B) = ((A \oplus B) \ominus B) \qquad \text{Closing} \tag{15}$$

Opening and closing operations can be used to eliminate noise, sampling errors, or thresholding errors. Opening eliminates white on black noise, and closing eliminates black on white noise. Noise elimination using opening or closing works like a size filter. It is assumed that the object of interest in the image is much larger than any random error. By performing successive iterations of opening or closing, all pixels corresponding to the error will be eliminated, whereas all pixels corresponding to the object of interest will remain unchanged. Examples of opening and closing operations are shown in Fig. 10.

A = {(-1,1),(0,1),(-1,-1),(0,-1)}

(a)



(b)

$$\begin{bmatrix} 1 & 1 & 0 \\ 0 & \text{org} & 0 \\ 1 & 1 & 0 \end{bmatrix}$$

(c)

Figure 9. Stelt representations: (a) list of vectors representation; (b) binary image representation (gray relevant pixel, black origin), (c) matrix representation, center pixel is the origin.

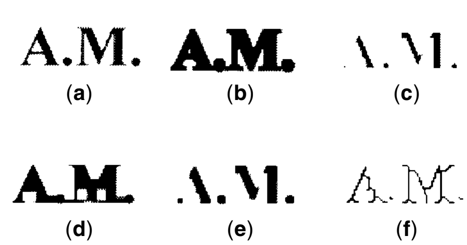


Figure 10. Examples of morphological operations: (a) original image, (b) dilation, (c) erosion, (d) closing, (e) opening, (f) skeletonization.

Thinning. A special kind of erosion operator is called thinning. Even though erosion normally causes an object to disappear with continued applications, the thinning operator continues to thin an object until the last pixel is reached. The result is a single pixel wide abstraction or *skeleton* of the original image. In thinning, the computer performs a sequence of erosions subject to the following constraints (2):

1. Pixels must be connected to at least one other pixel.
2. End pixels must not be eliminated to preserve length.

The result of a thinning operation is a skeleton one pixel thick. This skeleton is the core most representative of the object. The skeleton can then be used in pattern matching with a much smaller number of pixels required to describe the object. Thinning operations are often used as the first step in character recognition algorithms because different type styles generally alter the weight of the characters, not the general appearance. If the analysis is done on a skeleton of the character instead of the original image, there is a better chance of recognizing a wider array of type styles. Thinning algorithms are not generally suitable for a wide application in image analysis because they are very sensitive, especially when the object is composed of rectangular segments instead of long sweeping segments (7).

IMAGE UNDERSTANDING

Image understanding is concerned with the formation of hypothesis about the contents of a captured image. The image must be segmented and analyzed; then the objects within each segment of the image must be individually analyzed and identified. The effortless segmentation and recognition power of the human vision system has yet to be duplicated on a computer. The segmentation task parcels the image into several frames or regions of interest for further analysis. Although this can be a difficult step for complex scenes, for simple industrial scenes, segmentation can be performed quickly and easily. Object recognition algorithms encompass the entire process of analyzing the segmented image, extracting features, and matching with a knowledge base about the world. Object recognition itself covers many fields including artificial intelligence reasoning algorithms, pattern matching, and world representation.

The broad class of algorithms that encompass image understanding each have limitations. The sum of these limitations make image understanding a very difficult task. Even though humans can perform these tasks with apparent ease, computers cannot quickly and robustly perform general image understanding. There are distinct physical differences between human and computer processing that may account for the difference, but the lack of computer algorithms for general image understanding is not mitigated by the absence of fast, massively parallel computing. In general, image understanding algorithms work well in very simple, structured environments where all possible objects of interest are known, modelable and have distinct feature sets. The more general the system, the more unlikely that the image can be parsed correctly.

Segmentation

The segmentation includes two steps: processing and labeling. Processing was discussed earlier in the article in the context of basic image processing; labeling is the set of methods for grouping like pixel areas into objects. Common approaches to image processing for segmentation are thresholding, edge detection, and texture detection. These approaches divide the image into several regions that can then be analyzed individually by object recognition algorithms.

The choice of which processing algorithm to use depends on the scene under consideration and the type of segmentation the process requires. Simple thresholding is appropriate for very simple scenes like one or more nonoverlapping dark objects on a light background or a collection of radically different colored objects such as different crystals in a metal matrix. For more complicated scenes where objects have much the same color or intensity, edge detection or texture separation are more appropriate. The fundamentals of thresholding, texture analysis, and edge detection were described previously.

If the image is to be segmented using thresholding operations, it is normal to use an adaptive thresholding technique unless the image has high contrast and can be thresholded with a constant threshold value. The thresholded image should contain only two intensity values representing inside the object and outside the object. If there are multiple objects in the scene, it is often useful to map the objects to different gray-scale values. For example, if there are three objects in the scene, the thresholding operation should output four gray levels, one for each of the objects and another for the background (4).

Texture-based segmentation proceeds in much the same way as threshold-based segmentation. For each area of texture detected, the pixels are marked as belonging to that area. However, the marking may be difficult because the object recognition phase usually requires that the texture be left intact to aid identification. To avoid this problem, a copy of the image is segmented, whereas the original image and its texture information are left intact. The second image defines the regions of interest for the object recognition algorithm, whereas the first image contains the relevant feature information (3).

Regions that have been separated generally require postprocessing before they are supplied to the labeling system's input. Thresholding and texture segmentation algorithms may be corrupted by noise, which must be eliminated with a filter. Edge detectors leave regions that have boundaries greater than a single pixel thick and often have discontinuities, which must be eliminated. There are two general approaches for postprocessing edge detection boundaries: first, to interpret all the edges as geometric boundaries and, secondly, to create a digital boundary. If the object is represented by a set of geometric primitives such as line segments, the labeling algorithm can examine closed regions for both primitive and extended segments. Determining the direction of a boundary (what is inside and what is out) may require additional gradient direction information to be stored with the geometric primitive. If the objects are to be examined as a single pixel wide chain boundary, then the system should endeavor to close any gaps in the boundaries. If the chain ends anywhere but at the edge of the screen or at the starting point, then there is a gap that must be filled. Filling gaps can be accomplished by recursively searching all neighbors for an additional edge pixel and then drawing the shortest path line over the gap. This method of gap elimination usually works only for very small gaps in an uncluttered image.

Labeling

A simple and effective method of segmenting binary images is to examine the connectivity of pixels with their neighbors and label the connected sets.

Region labeling converts an image of class values into an image of connected components. Regions can be considered to be contiguous using either four-connectivity or eight-connectivity contiguity criteria. Four-connectivity involves checking only horizontally and vertically adjacent pixels when forming regions. Eight-connectivity involves additional checking for diagonally adjacent pixels. The most popular region-labeling algorithm is the run-tracking or run-length encoding method, which labels components by tracking runs of 1s (components' interior points). The procedure can be described in the following six steps:

1. On the first row of the picture that a 1 is encountered, each run of 1 is given a distinct label.
2. On the second (and succeeding) rows, runs of 1s are examined, and their positions are compared with the run on the previous row.
3. If the run p is adjacent (according to some definitions) to no runs on the previous row, p is given a new label.
4. If p is adjacent to just one run on the previous row, p is given the label of that run.
5. If p is adjacent to two or more runs on the previous row, p is given the lowest valued of their labels, but a note is also made of the fact that these labels all belong to the same component.
6. When the whole picture has been scanned in this way, the classes of equivalent labels are determined. If desired, the picture can be rescanned, and each label can be replaced by the lowest-valued equivalent label.

Using this method, individual components can be labeled. The problems with this kind of approach arise in practice mainly from the a priori assumption that parts must be held with string contrast to their surroundings and that they must not touch or overlap other workpieces. Of course, perfect images do not exist in real-world environments because noise in an image is unavoidable—this can easily cause misidentification of objects (3).

Recently, many different approaches have been proposed to increase the robustness of the segmentation by, for example, integrating segmentation and edge detection. Methods to integrate segmentation and edge detection can be classified as (1) knowledge-based methods, (2) pixel-wise Boolean methods, and (3) region refinement methods. Most of the recent work belongs to the third class. Among these, Pavlidis and Liow (8) describe a method to combine segments obtained by using a region-growing approach where edges between the regions are eliminated or modified based on contrast, gradient, and shape of the boundary. Haddon and Boyce (9) generate regions by partitioning the image co-occurrence matrix, and then refining them by relaxation using the edge information. Hedley and Yan (10) have suggested a histogram analysis to segment pixels with low spatial gradients, while edge pixels are later assigned to the nearest class computed from non-edge pixels. Chu and Aggarwal (11) presented an optimization method to integrate segmentation and edge maps obtained from several channels including visible and infrared, where user-specified weights and arbitrary mixing of region and edge maps are allowed. Saber et al. (7) recently presented a new, robust Bayesian region refinement method by region labeling. The proposed approach first computes an intermediate segmentation map, where labels form spatially contiguous clusters, called regions. Then, the class label of each region is optimized under a maximum a posteriori probability (MAP) criterion to merge regions whose boundaries are inconsistent with spatial edge locations.

Object Recognition

Object recognition is one of the fundamental aspects of machine vision. In any situation where the target is not distinct and known a priori (for example in an assembly line that only processes bolts), the individual elements of the scene must be identified. Many applications of computer vision are concerned wholly with the process of object recognition. For example, satellite images are scanned automatically for enemy installations, and medical images can be analyzed for evidence of tumors. In each of these tasks, the sole goal of the system is to identify an object in the image. Even tasks such as robot-guided vision require at least some image recognition to separate the area of interest out from the rest of the image.

Object recognition can be divided into several steps as shown in Fig. 11. There are many different approaches for each of these steps, sometimes based on extremely different paradigms (3).

Feature extraction creates a list of features for each object in the field of view. Objects generally cannot be identified from a list of pixels because the computational complexity of matching all pixels in the image to all known models is too high. Instead, the object is expressed as a set of features such as length perimeter, mean gray-scale level, and position in the image. These features are then passed to the a pattern-matching algorithm for further processing. The knowledge base is a set of objects that the object recognition algorithm knows how to identify. Objects in the knowledge base can be represented in many different ways depending on the applica-

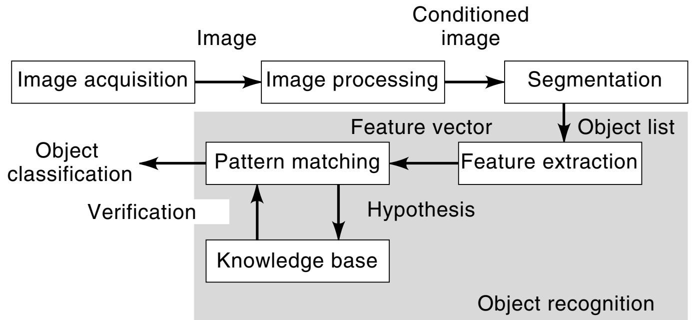


Figure 11. Block diagram of object recognition within the larger computer vision framework.

tion. The knowledge base provides the pattern matcher with a set of templates to compare to the current feature vector. The pattern matcher then generates a best guess or list of probable class types for the current feature vector based on known classes of objects stored in the knowledge base.

Variations on the basic object recognition strategy are possible. Feedback can be used if the list of probable class types is high, using tree-searching techniques and then feeding the results back through the classifier until a final hypothesis is reached. The feature extraction stage may also perform some preliminary object recognition by filtering out objects that do not satisfy broad criteria for the object of interest. For example, the feature extractor might return only square shapes if the recognition system were looking for a specific type of building in a satellite image, or the extraction step may return only large objects if the image is likely to be marred by small speckles of noise. Object recognition may also proceed in several steps, where first geometric primitives are identified and then the primitives are fed through another classifier that determines what object the primitives represent.

Feature Extraction. Feature selection and extraction are critical to the proper execution of an object recognition system. Features must be chosen such that the information encoded in the feature data is consistent for all objects of the class and sufficiently different from other classes to allow the pattern matcher to distinguish between different classes. Features are typically geometric, topological, or intensity based. Examples of different feature types follow:

- *Geometric.* Geometric properties are primarily used for encoding values such as size and position. Geometric properties are useful for eliminating noise on the basis of size and for applications that require the position of the object, such as vision-guided robotics.
- *Area.* Area is primarily used to determine size in images where the objects have known sizes and the scale varies (moving camera), or object size varies but the scale is constant (stationary camera). If both size and scale change, area measurements alone can be misleading.
- *Perimeter.* Perimeter can aid the determination of scale with area; however, because of the discrete nature of digital images, perimeter is very sensitive to noise and sampling errors.
- *Centroid.* The centroid or center of area is used to determine the center of the object. Note that the calculation occurs on the two-dimensional image before any three-dimensional reconstruction takes place. The measurement may not be valid for objects being observed from arbitrary angles in three space because of the different area profiles that are possible.
- *Topological.* Topological measurements are based on shape. Topological measurements are useful for classifying objects that have distinct shapes. For example, topological operators are good for distinguishing between nuts and bolts, but not oranges and grapefruit.
- *Number of Sides (for Polygon).* The number of sides can be used to distinguish between simple objects typically found in industrial environments. However, for curved objects, polygonal approximations can be imprecise; therefore, this measure is unreliable.
- *Fourier Descriptors.* Fourier descriptors describe the boundaries of an object using something akin to a two-dimensional Fourier expansion and describe the shape of a curved surface. However, the number of terms needed to describe objects with linear sides and sharp corners can be prohibitively large.
- *Euler Number.* Euler numbers are representations of the number of holes in an object. Euler numbers provide an excellent way of distinguishing between disks and washers. For homogeneous objects, the Euler number provides no distinguishing information.
- *Surface.* Surface characteristics are based on gray-scale, texture, and color values. In fact, surface features are direct results of the segmentation methods discussed previously. If objects have different surface characteristics, these measures can provide faster identification, essentially skipping the feature extraction step and proceeding directly to the object recognition stage.
- *Mean Intensity Value.* If the object can be separated based on gray-scale values, and its gray-scale distribution is approximately normal, then the mean gray-scale value of the distribution is a good way to characterize the object. Under variable lighting conditions, this can be a problem because the mean gray-scale value will change.
- *Texture Type.* If the object is characterized by a detectable texture, then texture may be used to identify an object.
- *Color Vector.* If color input is available, then the object can be expressed in terms of mean RGB (red, green, blue) or HSI (hue, saturation, intensity) values. Color values, like intensity, are susceptible to changes in lighting conditions.

Objects may also be described as geometric primitives, such as linked line, usually as the result of an edge detection algorithm. These representations can be considered feature array, like describing a quadrilateral as four line segments, or they could be processed to find different or additional features, such as counting the number of sides to distinguish between triangles and rectangles.

Features are extracted from the image after segmentation, and the format of the features depend on the type of segmentation performed. If a region-based segmentation is used, then geometric and topological features are expressed as pixel counts. If a geometric boundary representation was derived, features such as area and perimeter will be calculated using geometric relationships between the different features. The type of representation is generally not important as long as the expressions are consistent, and the type of representation is taken into account in the pattern-matching algorithm.

Feature selection is an important step in object recognition. There are several constraints and requirements for good features: features must be simple to extract given the hardware and target objects; features must describe the object reliably; and features must differentiate the object from other possible objects and the background of the image.

The question of feature selection, and especially automated feature selection, has received a great deal of attention. Because most feature selection processes require more than three features to classify all the objects correctly, it is difficult to graph the relationships between features and objects, especially when the object can be viewed from many different angles. For example, looking for a triangular area would help locate the roof of a house for a side view but would not work from a top view where the roof appears rectangular. In addition, it is necessary to consider the joint distributions of different features because additional features could offer more or less information depending on the features already selected. For example, if a triangle is characterized by the positions of each of its vertices, adding the length of each side to the feature vector would add less information to the system than adding the color of the triangle. This is because the lengths of each side are functions of the positions of the vertices, whereas the color of the triangle is completely independent (3).

Selecting features is usually a pruning exercise based on designer knowledge and statistical measurements. The designer selects a very large number of possible features to measure and takes a statistically large sample of data. Graphing the probability density functions of the measured features shows the independent behavior of each feature. If the distribution of the features is unique enough, it may be possible to select a group of features by inspection. Usually the problem is not so simple, and several statistical measures of the feature data must be taken to reduce the feature set to a manageable size. The data gathered for evaluating the performance of each feature should be gathered in bins corresponding to the class of object that was being analyzed, making the generation of the statistical measurements simpler. Statistical measures that should be considered are the mean value of the feature for each class, the feature variance for each class, the feature covariance with other features on the same class, and the ability of the feature to distinguish between classes. Ideally, features should have a mean corresponding to a single Gaussian peak, low variance, zero covariance, and large class separation (4).

World Representation

The techniques of representing the world as computer models are more properly discussed in the context of computer-aided design (CAD) systems. However, the way the world is represented in the computer does have an impact on how the designer must approach the construction of a vision system. Therefore, this section will deal with the basic properties of representation and how they affect image understanding. The reader is referred to the sections in this encyclopedia describing CAD for details on precise implementations of these techniques.

There are two basic types of world representation: object centered and view centered. In an object-centered technique, a physical model of the real world is created in the computer, and that model is compared with the current image. Object-centered representation requires a transform from object space to feature space or from feature space to object space in order for the comparisons to be made. View-centered representations are direct records of the expected feature set. They are more efficient than object-centered representations because the measured data can be compared directly with the model without any additional transforms. View-centered representations are less general than object-centered models because the transform between object and features is implicitly recorded and assumed constant (4).

There are several different techniques for recording object-based models. For two-dimensional objects, or objects on view from a fixed perspective, a simple two-dimensional line drawing is usually sufficient. The drawing may contain other values associated with it such as Euler number and area if these values are required for the application. If the object is three-dimensional and can be viewed from several possible angles, a three-dimensional representation should be chosen. Three commonly used three-dimensional representations are constructive solid geometry, boundary surface representation, and voxel representation. Constructive solid geometry (CSG) relies on a set of primitive shapes such as cylinders, cubes, and spheres to create larger, more complicated objects. The primitives are combined using simple arithmetic operations such as addition and subtraction and logical set operations like union and intersection. CSG can create very compact and intuitive representations of simple objects, but it is quite computationally expensive for secondary features such as surface area or projections. CSG models are best if very precise models of manufactured items are required. A surface model can also be used to represent a three-dimensional solid. The surface model is composed of all the surfaces of an object and a normal vector to describe what is inside the object and what is outside the object. By convention, the normal vector points away from the object. Surface models require more storage space than CSG objects but are slightly more computationally efficient when calculating geometric features. Surface models are best for simple objects with rectilinear edges. Finally for complex objects, voxel representations can be used. A voxel is a small cube and is the three-dimensional equivalent of a pixel. Voxels approximate objects as collections of small cubes; the smaller the cube, the better the approximation. Voxels are the least accurate and require the most space of the three-dimensional representation methods, but they are very effective at representing natural objects with arbitrarily curved surfaces. The type of representation chosen depends on the target type, the available hardware, and the degree of accuracy required (12).

Object-centered models require transforms from model space to feature space. Usually this begins with computing the viewing transform and then calculating the features based on the two-dimensional projection of the model. The computations involve expressing the object not as it is stored but as it appears on the screen. This may require additional interpretation of the model because the projection may cause some parts of the object to appear as holes or unconnected to the main body of the object. Generally, the resulting model in image space must be analyzed using the same methods as the current image of the external world.

Feature-centered models assume that the transform between object and features is constant and, therefore, does not need to be computed. The object stored is a feature vector and can be directly compared to the measured features using a pattern-matching algorithm. The absence of a model-to-feature transform makes the feature-based model faster than the object-based model; however, the feature-based model is limited in scope. If the transform between the features and the object change, then the feature-centered model can create incorrect or erroneous results because the transform is inherently part of the model and assumed constant.

Pattern Matching

The process of deciding what object the image contains is performed by a pattern-matching algorithm. Pattern-matching algorithms are the mathematical interface between the numerical feature data and the world representation. Pattern-matching algorithms can be based on many different mathematical disciplines, fixed or adaptive, human-derived or numerically derived. Although it is outside the scope of this work to present a full discourse on pattern-matching and recognition algorithms, the general properties of several algorithms will be presented in the context of object recognition for computer vision.

There are two fundamental techniques for pattern recognition: those based on human models and a priori information and those based on adaptive numerical approximations. Model-based techniques are generally statistical, whereas adaptive techniques can be based on statistics, fuzzy logic, or neural networks. Adaptive techniques are best for applications with a large input space and uncertain feature-class mappings. Adaptive classifiers do not work as well for complex models involving relationships between the features as well as the simple numerical feature data because it is hard to derive interfeature relations numerically. The role of relations between features in models is discussed in the section on knowledge-based techniques. Model-based approaches work well with knowledge-based techniques because they allow the user to encode all types of a priori information, not just numerical feature data. Model-based techniques are not as appropriate for systems that are difficult to model, usually involving a large ($>$ 10 dimensional) feature space with significant correlation between different features for object classes (3).

Model-based methods are essentially statistical in nature. The system is modeled by a set of distributions, mapping the feature data to a given class. Ideally, it will be represented by an impulse function, all instances of the class having exactly the same features with negligible error due to the image acquisition system. Because this is almost never the case, statistical models of the feature-class mapping must be derived. Generally, if the only variations are caused by errors in the image acquisition or minor differences among objects in a class, the model should have a normal distribution. This normal distribution can be considered a cluster by assuming that all objects outside a certain probability threshold are not part of the class. If each class is considered a cluster, then several operators can be used to separate them and identify new classes.

The simplest operator is the nearest neighbor operator. If the system is classified by N classes, then each class is recorded as a typical or mean feature set. This feature set is usually derived as the mean of each feature for the class. When the classifier is run, the generated feature set is compared to each of the mean feature sets for each class using a distance metric like the Euclidean distance:

$$d = \sqrt{\sum (f_{\mathrm{m}} - f_{\mathrm{ci}})^2} \tag{15a}$$

The current object under observation is assigned to the set that is closest as defined by the distance metric. The nearest neighbor operator can be enhanced by allowing the possibility of an unknown class. If the distance to all classes is above a threshold, then the algorithm can return the class as unknown. This can solve the problem of misinterpretation of poor data samples. The nearest neighbor operator is a good operator for simple problems because it is easy to create and fairly robust. However, the algorithm assumes that all the classes are distinct and that there are no outliers that would fall into another class. The nearest neighbor class also assumes that an object can be reliably classified by the mean value of its feature set taken over a large number of samples. Even though this is true for objects with normal distributions, it is not necessarily true for classes where the distribution is not normal (3).

A second class of statistical classifiers is a feature space partitioning. This is somewhat similar to the nearest neighbor operator operation in that it assumes that a priori models of the distributions of the classes are known, and it allows more general boundaries to be defined. The nearest neighbor approach partitions feature space into hyperspheres centered on the class mean if the Euclidean distance is used as a metric. If the hyperspheres overlap, then a more specific partition of feature space using hypercubes, plains, or conics might be more reasonable. Every feature vector input during execution is viewed as a point in N-dimensional feature space and is assigned based on the area of N space it occupies. Even though this approach is mathematically valid, it requires the user to partition an N-dimensional feature space; it is possible for up to three dimensions but rapidly becomes a very difficult problem. The second assumption is that each set is perfectly separable, which may not be valid for some identification problems. For example, if the vision system identified different types of apples, there would be significant overlap between Spartan and Macintosh apple classes.

Bayesian statistics provides a robust tool for estimating the membership of a sample into a class of objects. Bayesian statistics does not require that the feature space be completely partitioned; instead, it assigns probabilities of set membership and allows the user to select the most appropriate response, Bayes's theorem states that

$$p(C_i|x_1, x_2, \ldots, x_n) = \frac{p(x_1, x_2, \ldots, x_n|C_i)P(C_i)}{\sum_{i=1}^{m} p(x_1, x_2, \ldots, x_n|C_i)P(C_i)} \tag{16}$$

or that the probability that the object represented by the current feature vector is of class C_i is the conditional probability that the feature vector is a member of C_i divided by the sum over all possibilities that the feature vector belongs to any class C_i over all possible m classes. The best class is chosen by minimizing a risk function. The risk function is a weighted sum that encodes the danger of misdiagnosis, essentially forcing the probabilities from the previous expression to be higher if the current class C_i has a high risk associated with an incorrect positive result (4). For example, there is very litte inherent risk identifying different sorts of apples; if one Spartan is included with one thousand Macintoshes, no one will suffer. However, if a lab blood sample is diagnosed clean when the patient actually has a disease, the patient could become very sick because the appropriate medication was not administered. If the risk factor is expressed as a weight vector, the Bayesian risk can be written as

$$R(C_i|x_1, x_2, \ldots, x_n) = \sum_{j=1}^{m} w_{ij} p(C_j|x_1, x_2, \ldots, x_n) \tag{17}$$

where w_{ij} is the weighted risk factor for the expression. The best classification is the class where the risk is minimum. An additional class, *unidentified* can be added for situations where the risk is greater than a threshold. This class is particularly useful for avoiding false positives in high-risk situations. If an object is not identifiable, operation can be suspended until a human operator can identify the item manually and resume normal operation.

Even though model-based approaches can be quite accurate for simple systems, complex systems can be difficult to model and almost impossible to partition. Many adaptive techniques have been suggested for classifiers that use numerical techniques to create the best approximations for the feature class mappings. These classifiers come in two general categories, classifiers that make assumptions about the shape of the probability density functions (pdf) and classifiers that derive the shape of the probability density function. Adaptive algorithms are usually created from a training set of data and then updated during regular processing.

A very simple adaptive technique called maximum-likelihood estimation assumes that the pdf is normal and that the mean and standard deviation of the distribution are the mean and standard deviation of the training set. As execution commences, a running mean and standard deviation are kept and update the current mean and standard deviation accordingly. Maximum-likelihood can be coupled with the neighborhood operator to provide a simple adaptive algorithm. If a Bayesian classifier is used, then Bayesian updating techniques can be used to find the optimal mean value for the conditional pdf of the function. The Bayesian technique is much the same as the maximum-likelihood technique, except that the system is given an a priori estimate of the mean and standard deviation. The estimate of the standard deviation determines the degree of change of the mean due to the update rule. If the initial function width is guessed to be large, the mean of the pdf will tend toward the mean of the training set. If the standard deviation of the system is small, then the update rule does not change the a priori estimate of the mean by a large amount. The update function is given by

$$p(x|X) = \int_{-\infty}^{\infty} p(x|u)p(u|X)du$$

where $p(x|X)$ is the updated pdf, $p(x|u)$ is the a priori pdf, and $p(u|X)$ is the sample training set pdf.

Research into artificial neural networks has led to new types of adaptive classifiers where the shape as well as the parameters of the feature-class mapping can be determined computationally. The reader is referred to Ref. 13 for details of the classifiers discussed here as well as a detailed account of neural classifier operation. For the purposes of this article, two neural networks will be discussed—self-organizing feature maps and backpropagation networks.

Self-organizing feature maps (SOFMs) are an example of unsupervised neural networks. The neural network provides output based solely on clustering the input data; the user does not need to supply the desired output. The SOFM network takes the feature vector as input and returns the class at output. The SOFM is a two-layer network, with n input nodes for the feature vector and m output nodes for the classification. The SOFM network essentially performs the same calculations as a nearest neighbor classifier except that it derives the best representation itself. This is appropriate when the distribution is not normal and is difficult to model from the feature data. Like the nearest neighbor algorithm, the SOFM subdivides the feature space into hyperspheres and creates the best fit for the data for a set of hyperspheres (13).

The backpropagation neural network is an example of a supervised neural network. During training, the desired output for the classifier must be fed back into the neural network so that the network can be properly trained using a gradient decent algorithm. The backpropagation network is typically three layers—an input layer, a hidden layer, and an output layer. The input and output layers are generally linear, and the hidden layer applies a sigmoidal function to its inputs. The backpropagation network provides the most general form of adaptive pattern recognition described in this article because it expresses the boundaries as the weighted sum of sigmoidal functions, which means that the partitioning of the feature space can have a very general shape. It is appropriate to use backpropagation networks when the correct response can easily be added to the training set beforehand, and the classes are not clearly differentiable in any way (13).

Knowledge-Based Vision

Traditional computer vision systems attempt to recognize static objects and their dynamic behavior using bottom-up, data-driven processing with minimal use of prior knowledge. However, such systems are bound to fail for complex domains as the information in the images alone is insufficient for detailed interpretation or understanding of the objects and events. Knowledge-based vision research relies primarily on scene context to overcome this kind of uncertainty. Incorporating knowledge into a computer vision system requires choice of formalism in which to express this knowledge. The three principal approaches used are based on (1) formal logic, (2) semantic networks, and (3) production systems (3).

Knowledge-based systems are typically domain dependent. An example is used here to illustrate how they can be developed. The system is developed for image segmentation based on the design of a rule-based expert system (14). A rule has the following format:

CONDITION.AND.CONDITION . . .
. . . AND.CONDITION ACTIONS

The left-hand side is composed of a set of CONDITIONS evaluated on the image data. The ACTIONS on the right-hand side specify particular modifications to the data. The logical ANDs indicate that the action of the rules will be performed only if *all* its conditions are satisfied. A few samples of the rules stored in the rule base are given as follows:

Rule (908):
If (1) The REGION SIZE is NOT LOW
(2) REGION is BISECTED BY LINE
(3) The LINE LENGTH is NOT LOW
(4) The LINE AVERAGE GRADIENT is HIGH
Then: (1) SPLIT the REGION at LINES

Rule (1502):
If (1) The LINE END point is OPEN
(2) The DISTANCE to the LINE IN FRONT is NOT HIGH
(3) The LINE AVERAGE GRADIENT is NOT LOW
Then: JOIN the LINES by FORWARD expansion

The rule-based model is composed of three levels of production rules. At the first level, the knowledge rules encode the information about properties of regions, lines, and areas in the form of sets of situation-action pairs. The conditions are joined by AND so that, when a specific situation occurs within the image, all the conditions will be met. In this case, a match is said to have occurred, and the rule fires. The rule action is then executed. The second level of rules in the knowledge base contains the control rules. There are two categories of control rules. The first category of rules are responsible for finding the next data entry to be considered. They have actions that bring to the attention of the system a next region, a next line, or an entire area. Using these rules, a strategy for visiting regions and lines within a region of interest can be defined and executed. The second category of rules are referred to as metarules. Their actions do not modify the data in the knowledge base. Instead, the metarules alter the matching order of different knowledge rule sets. Thus the first category of rules define the method by which data are selected for processing; the metarules specify the order in which the rule sets are matched.

This system introduces the rule-based approach to the image segmentation problem. The approach employs domain-independent knowledge in an explicit form. Knowledge is separated from processing modules by coding it into production rules, which are stored in the knowledge base.

A number of knowledge-based vision system have been recently reported. For example, Strat and Fischler (15,16) combine many simple vision procedures that analyze color, stereo, and range images with relevant contextual knowledge to achieve reliable recognition. There are many other types of contextual knowledge such as functional context (17), where attributes such as shape are used to infer the functional role of the object and direct the visual processing (18). Another type of context, which is particularly relevant to multimodal and multimedia systems, is linguistic context (19,20). In addition, task context is an important source of control for the visual processing (21–23). The role of context, then, is central to visual interpretation and understanding, and representing context in an appropriate way to improve the effectiveness and efficiency of visual reasoning is a key issue in the field.

BIBLIOGRAPHY

1. B. Jähne, *Practical Handbook on Image Processing for Scientific Applications.* New York: CRC Press, 1997.
2. J. C. Russ, *The Image Processing Handbook: Second Edition.* Boca Raton, FL: CRC Press, 1995.
3. D. H. Ballard and C. M. Brown, *Computer Vision.* Englewood Cliffs, NJ: Prentice Hall, 1982.
4. R. Jain, K. Rangachar, and B. G. Schunk, *Machine Vision.* New York: McGraw-Hill, 1995.
5. R. Klette and P. Zamperoni, *Handbook of Image Processing Operators.* New York: Wiley, 1996.
6. O. Faugeras, *Three Dimensional Computer Vision: A Geometric Viewpoint.* Cambridge, MA: MIT Press, 1993.
7. R. C. Vogt, *Automatic Generation of Set Recognition Algorithms,* New York: Springer-Verlag, 1989.
8. T. Pavlidis and Y. T. Liow, Integrating region growing and edge detection, *IEEE Trans. Pattern Anal. Mach. Intell.,* **PAMI-12**: 225–233, 1990.
9. J. F. Haddon and J. F. Boyce, Image segmentation by unifying region and boundary information, *IEEE Trans. Pattern Anal. Mach. Intell.,* **PAMI-12**: 929–948, 1990.
10. M. Hedley and H. Yan, Segmentation of color images using spatial and color space information, *Electron Imaging,* **1**: 374–380, 1992.
11. C. C. Chu and J. K. Aggarwal, The integration of image segmentation maps using region and edge information, *IEEE Trans Pattern Anal. Mach. Intell.,* **PAMI-15**: 1241–1252, 1993.
12. I. Zeid, *CAD / CAM Theory and Practice.* New York. McGraw-Hill, 1991.
13. S. Haykin, *Neural Networks A Comprehensive Foundation.* New York: Macmillan College Publishing Co., 1994.
14. A. M. Nazif and M. D. Levine, Low level image segmentation: An expert system, *IEEE Trans. Pattern Anal. Mach Intell.,* **PAMI-6** (5): 1994.
15. T. M. Strat and M. A. Fischler, Context-based vision: Recognizing objects using both 2D and 3D imagery, *IEEE Trans. Pattern Anal. Mach Intell.,* **13**: 1050–1065, 1991.
16. T. M. Strat and M. A. Fischler, The role of context in computer vision. *Workshop on Context-based Vision.* Piscataway, NJ: IEEE Press, 1995.
17. L. Stark and K. Bowyer, Functional context in vision. *Workshop on Context-Based Vision,* IEEE Press, NJ: 1995.
18. L. Birnbaum, M. Brand, and P. Cooper, Looking for trouble: Using causal semantics. *Int. Conf. Computer Vision,* Berlin, Germany, 1993.
19. G. Socher et al., Talking about 3D scenes: Integration of image and speech understanding in a hybrid distributed system. *Int. Conf. Image Processing,* Lausanne, Switzerland, 1996.
20. Y. Shoham, *Reasoning about Change: Time and Causation from the Standpoint of Artificial Intelligence.* Cambridge: MIT Press, 1988.
21. R. K. Srihari, Linguistic context in vision, *Workshop on Context-based Vision.* Piscataway, NJ: IEEE Press, 1995.
22. H. Buxton and S. Gong, Visual surveillance in a dynamic and uncertain world, *Artificial Intelligence,* **78**: 371–405, 1995.
23. J. L. Crowley and H. Christensen, *Vision as Process.* Berlin: Springer-Verlag, 1993.

Q. M. Jonathan Wu
Kevin Stanley
National Research Council of Canada
Farzin Deravi
University of Wales
Dewey Liew
National Research Council of Canada

COMPUTER VISION. See Camera calibration for image processing; Image processing.

COMPUTER VISION, BINOCULAR. See Stereo image processing.

COMPUTER VISION FOR ROBOTS. See Machine vision for robotics and inspection.

COMPUTING, BIOLOGY. See Biology computing.

COMPUTING, HOME. See Home computing services.

COMPUTING IN ENVIRONMENTAL SCIENCE. See Environmental science computing.

COMPUTING, NETWORK-BASED. See Network computing.

COMPUTING, PERSONAL. See PERSONAL COMPUTING.
CONCEPTUAL DATA MODELS. See DATABASE MODELS.